IN THE CLAIMS:

The text of all pending claims, (including withdrawn claims) is set forth below. Cancelled and not entered claims are indicated with claim number and status only. The claims as listed below show added text with underlining and deleted text with ~~strikethrough~~. The status of each claim is indicated with one of (original), (currently amended), (cancelled), (withdrawn), (new), (previously presented), or (not entered).

1. (Original) A load-module creating method that creates a load module for a program that has plural processes, each process of the program being executed by one processor out of plural processors, the load-module creating method comprising:

determining if, from amongst the plural processes of the program, at least two processes read the same data included in the program;

affixing specific identification information to the data that is determined to be read by at least two processes of the program; and

forming a non-cacheable area in a memory space where all the data to which the identification information is affixed are kept.

2. (Original) The load-module creating method according to claim 1, wherein the affixing includes affixing a specific prefix, as the identification information, to the data.

3. (Original) The load-module creating method according to claim 1, wherein the affixing includes affixing information that specifies the section to which the data belongs, as the identification information, to the data.

4. (Original) A load-module creating method that creates a load module for a program that has plural processes, each process of the program being executed by one processor out of plural processors, the load-module creating method comprising:

determining if, from amongst the plural processes of the program, at least two processes read the same data included in the program; and

affixing a cache-invalidate operation instruction to the data that is determined to be read by at least two processes of the program.

5. (Original) The load-module creating method according to claim 4, wherein the

affixing includes inserting an invalidate instruction for the data just before a load instruction for the data.

6. (Original) A load-module creating method that creates load modules for a first program executed by a first processor and a second program executed by a second processor, the load-module creating method comprising:

building a first set of memory areas by linking a first set of objects of the first program executed by the first processor;

computing an address for a first symbol in the first program, based on address of the first set of memory areas at the building of the first set of memory areas;

building a second set of memory areas by linking a second set of objects of the second program;

computing an address for a second symbol in the second program, based on the address of the second set of memory areas at the building of the second set of memory areas; and

computing, based on the address of the second symbol, the address of the first symbol whose address is not computed.

7. (Original) The load-module creating method according to claim 6, wherein if there is an address of the first symbol that was not resolved, then the non-resolved address is computed by determining the offset of the address of the second symbol from an initial address of the second symbol in the second set of memory areas.

8. (Original) The load-module creating method according to claim 6, wherein the building of the first memory space includes forming a first memory area that can be accessed only by the first processor and a second memory area that can accessed by the first processor as well as the second processor.

9. (Original) The load-module creating method according to claim 8, wherein the second memory area is located in a memory of a processor element of the second processor.

10. (Original) A computer program that causes a computer to execute a method that creates a load module for a program that has plural processes, each process of the program being executed by one processor out of plural processors, the computer program comprising:

determining if, from amongst the plural processes of the program, at least two processes read the same data included in the program;

affixing specific identification information to the data that is determined to be read by at least two processes of the program; and

forming a non-cacheable area in a memory space where all the data, to which the identification information is affixed, are kept.

11. (Original) The computer program according to claim 10, wherein the affixing includes affixing a specific prefix, as the identification information, to the data.

12. (Original) The computer program according to claim 10, wherein the affixing includes affixing an information that specifies the section to which the data belongs, as the identification information, to the data.

13. (Original) A computer program that causes a computer to execute a method that creates a load module for a program that has plural processes, each process of the program being executed by one processor out of plural processors, the computer program comprising:

determining if, from amongst the plural processes of the program, at least two processes read the same data included in the program; and

affixing a cache-invalidate operation instruction to the data that is determined to be read by at least two processes of the program.

14. (Original) The computer program according to claim 13, wherein the affixing includes inserting an invalidate instruction for the data just before a load instruction for the data.

15. (Original) A computer program that causes a computer to execute a method that creates load modules for a first program executed by a first processor and a second program executed by a second processor, the computer program comprising:

building a first set of memory areas by linking a first set of objects of the first program executed by the first processor;

computing an address for a first symbol in the first program, based on address of the first set of memory areas built;

building a second set of memory areas by linking a second set of objects of the second program;

computing an address for a second symbol in the second program, based on the address of the second set of memory areas built; and

computing, based on the address of the second symbol, the address of the first symbol whose address is not computed.

16. (Original) The computer program according to claim 15, wherein if there is an address of the first symbol that was not resolved, then the non-resolved address is computed by determining the offset of the address of the second symbol from an initial address of the second symbol in the second set of memory areas.

17. (Original) The computer program according to claim 16, wherein the building of the first memory space includes forming a first memory area that can be accessed only by the first processor and a second memory area that can accessed by the first processor as well as the second processor.

18. (Original) The computer program according to claim 17, wherein the second memory area is located in a memory of a processor element of the second processor.

19. (Original) A load-module creating apparatus that creates a load module for a program that has plural processes, each process of the program being executed by one processor out of plural processors, the load-module creating apparatus comprising:

a shared data determining unit that determines if, from amongst the plural processes of the program, at least two processes read the same data included in the program;

an identification information affixing unit that affixes specific identification information to the data that is determined to be read by at least two processes of the program by the shared data determining unit; and

a shared data area forming unit that forms a non-cacheable area in a memory space where all the data to which the identification information is affixed by the identification information affixing unit are kept.

20. (Original) The load-module creating apparatus according to claim 19, wherein the identification information affixing unit affixes a specific prefix, as the identification information, to the data.

21. (Original) The load-module creating apparatus according to claim 19, wherein the identification information affixing unit affixes information that specifies the section to which the data belongs, as the identification information, to the data.

22. (Original) A load-module creating apparatus that creates a load module for a program that has plural processes, each process of the program being executed by one processor out of plural processors, the load-module creating apparatus comprising, a shared data determining unit that determines, if from amongst the plural processes of the program, at least two processes read the same data included in the program; and

a cache-invalidate operation instruction affixing unit that affixes a cache-invalidate operation instruction to the data that is determined to be read by at least two processes of the program by the shared data determining unit.

23. (Original) The load-module creating apparatus according to claim 22, wherein the cache-invalidate operation instruction affixing unit inserts an invalidate instruction for the data just before a load instruction for the data.

24. (Original) A load-module creating apparatus that creates load modules for a first program executed by a first processor and a second program executed by a second processor, the load-module creating apparatus comprising:

a first memory space building unit that builds a first set of memory areas by linking a first set of objects of the first program;

a first intra-memory address resolution unit that computes an address for a first symbol in the first program, based on the address of the first set of memory areas formed by the first memory space building unit;

a second memory space building unit that builds a second set of memory areas by linking a second set of objects of the second program;

a second intra-memory address resolution unit that computes an address for a second symbol in the second program, based on the address of the second set of memory areas formed by the second memory space building unit; and

an inter-memory space address resolution unit that computes, based on the address of the second symbol computed by the second intra-memory address resolution unit, the address of the first symbol whose address was not resolved by the first intra-memory address resolution unit.

25. (Original) The load-module creating apparatus according to claim 24, wherein if there is an address of the first symbol that was not resolved, the inter-memory space address resolution unit computes the non-resolved address by determining the offset of the address of the second symbol computed by the second intra-memory address resolution unit from an initial address of the second symbol in the second set of memory areas formed by the second memory space building unit.

26. (Original) The load-module creating apparatus according to claim 24, wherein the first memory space building unit forms a first memory area that can be accessed only by the first processor and a second memory area that can be accessed by the first processor as well as the second processor.

27. (Original) The load-module creating apparatus according to claim 26, wherein the second memory area is located in a memory of a processor element of the second processor.